Minimal Objectification and Maximal Unnesting in PSOA RuleML

Gen Zou, Harold Boley

Faculty of Computer Science, University of New Brunswick, Canada

The 10th International Web Rule Symposium, RuleML 2016
Stony Brook University, 6-9 July 2016

Outline

Outline

PSOA RuleML

- Object-relational Web rule language that generalizes **relationships** (e.g., in FOL, LP) and **frames** (e.g., in RDF, N3) into **p**ositional-**s**lotted **o**bject-**a**pplicative (**psoa**) terms

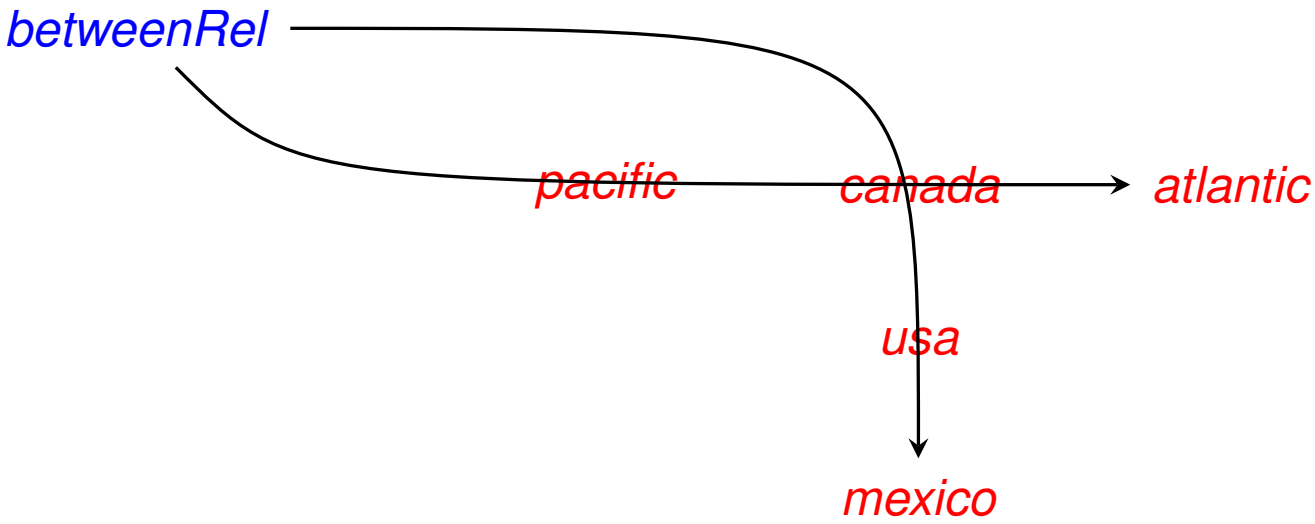
General cases (multi-tuple), where "#" means "member of":

Oidless: $f([t_{1,1} \ldots t_{1,n_1}] \ldots [t_{m,1} \ldots t_{m,n_m}]\ p_1 \texttt{->} v_1 \ldots p_k \texttt{->} v_k)$

Oidful: $o\#f([t_{1,1} \ldots t_{1,n_1}] \ldots [t_{m,1} \ldots t_{m,n_m}]\ p_1 \texttt{->} v_1 \ldots p_k \texttt{->} v_k)$

- Oidless psoa terms are interpreted as **atoms** (i.e., predicate applications) on the top level and as **expressions** (i.e., function applications) when embedded in another term
- Oidful psoa terms are interpreted as atoms both on the top-level and when embedded
- **Interchangeable** use of oidless and oidful **atoms** is enabled via **objectification** transformation
- Embedded oidful atoms can be extracted via **unnesting**

Special Case of Oidless Atom – Relationship



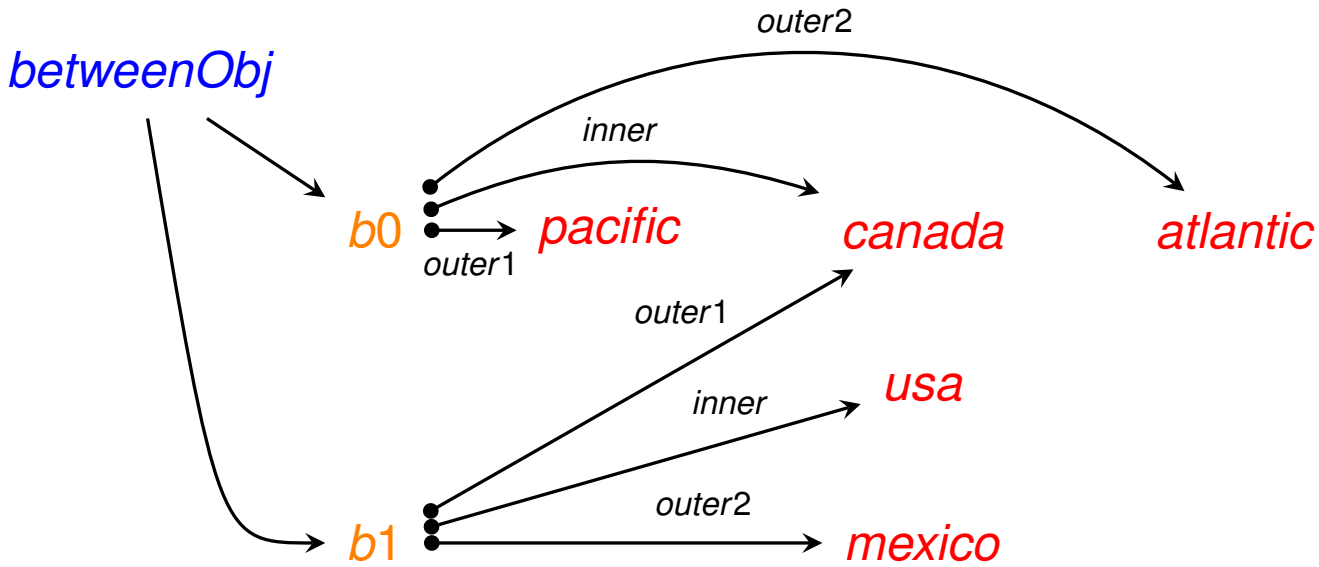
Directed hyperarcs cut through intermediate nodes (cf. Grailog)

Facts

Surrounding brackets of tuples can be omitted for single-tuple psoa terms

betweenRel(*pacific canada atlantic*)
betweenRel(*canada usa mexico*)

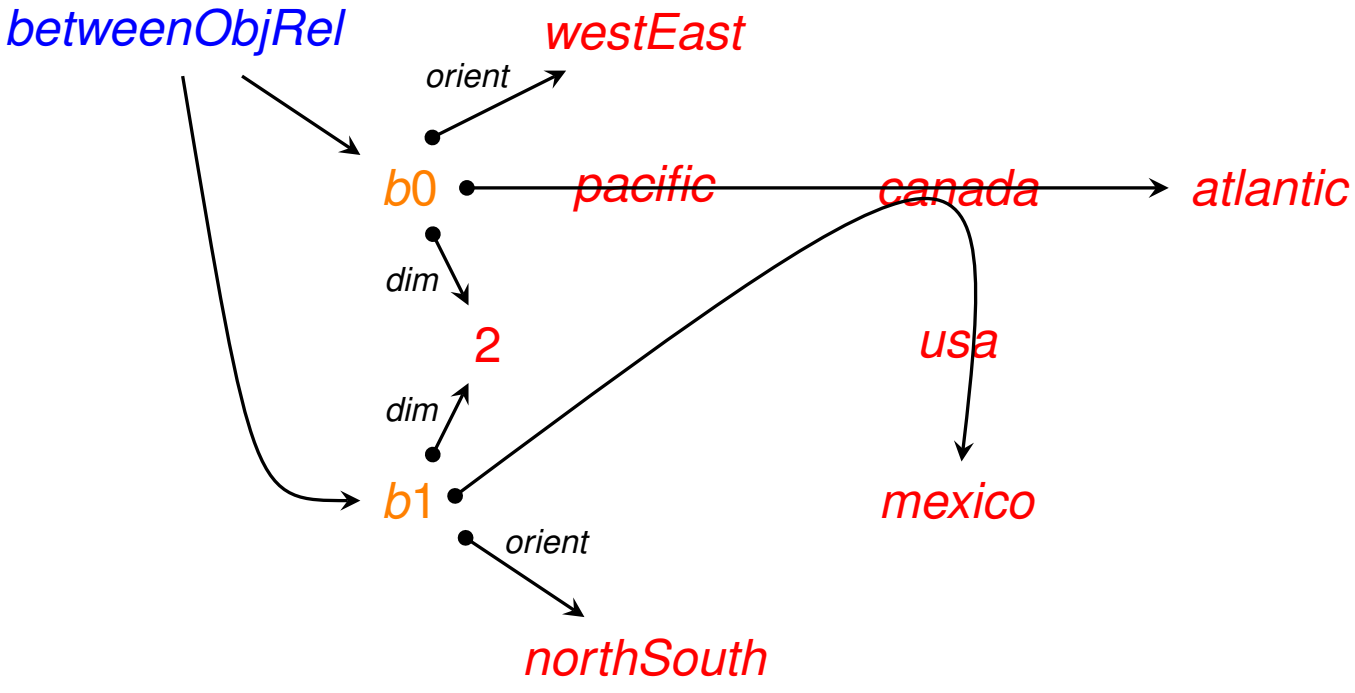
Special Case of Oidful Atom – Frame



Facts

$b0\#betweenObj(outer1 \rightarrow pacific \quad inner \rightarrow canada \quad outer2 \rightarrow atlantic)$
$b1\#betweenObj(outer1 \rightarrow canada \quad inner \rightarrow usa \quad outer2 \rightarrow mexico)$

Special Case of Oidful Atom – Shelframe (Enriched)



Facts

$b0\#betweenObjRel(pacific\ canada\ atlantic \quad dim \rightarrow 2 \quad orient \rightarrow westEast)$
$b1\#betweenObjRel(canada\ usa\ mexico \quad dim \rightarrow 2 \quad orient \rightarrow northsouth)$

PSOATransRun 1.1

- **Efficient** reasoning in PSOA RuleML enabled
  - All forms of psoa atoms used as facts or in rules
  - Embedded oidful psoa atoms
  - (OID-)head-existential rules
  - Equality in the body, restricted to unification and external-function evaluation
  - Subclass formulas for 'ABox' reasoning only
  - Built-in arithmetic functions
- **Released** in Java source form and as an executable jar file. Downloadable from: `http://wiki.ruleml.org/index.php/PSOA_RuleML#PSOATransRun`
- Includes a composition of translator PSOA2Prolog and well-known XSB Prolog engine

Outline

Static vs. Dynamic Objectification of Atoms – Preview

KB: `_work(_Kate _Rho4biz "Director")`

Query: `?O#_work(?P ?C ?J)`

(Users can pose oidless/oidful queries regardless of whether the underlying KB clauses have OIDs or not)

Static: Generate explicit OID
(transform above **KB** ground atom, use **query** unchanged):

- Undifferentiated (using existential OID variable):
  `Exists ?1 (?1#_work(_Kate _Rho4biz "Director"))`
- Differentiated (using fresh OID constant):
  `_1#_work(_Kate _Rho4biz "Director")`

Dynamic: Virtualize with '`_oidcons`' function and equality '='
(keep above **KB** unchanged, transform **query** atom):

```
And(_work(?P ?C ?J)
    ?O=_oidcons(_work ?P ?C ?J))
```

Objectification – Semantics

- Old semantics
  - Can only interpret an oidless psoa term after applying static objectification
  - Causes reasoning overhead for an atom whose predicate in the KB clauses is used only as a Prolog-like relation, e.g. does not occur with an OID or slots
  - Cannot deal with an expression term – which returns an arbitrary value – since giving it an OID would make the function act as the class of the OID and lead to a truth value
- New semantics
  - Allow direct interpretation and truth evaluation of oidless psoa terms
  - Add **objectification restriction**

$$TVal_{\mathcal{I}}(\texttt{f(...)}) = \mathbf{t}$$

if and only if

$$TVal_{\mathcal{I}}(\texttt{Exists ?O (?O\#f(...))}) = \mathbf{t}$$

Objectification Transformation Systematics

- Objectification transformation realizes the objectification restriction by transforming KBs and queries such that entailments can be established under the semantics in which the restriction is excluded (cf. Def. 2)
- Systematics of objectification transformation of KBs/queries
  - **Static**: generate explicit OIDs for **all** of the KB's oidless atoms
    - **Undifferentiated**: uniformly transforms oidless atoms everywhere (cf. Def. 4)
    - **Differentiated**: transforms oidless atoms based on their occurrences (cf. Def. 5)
  - **Static/Dynamic** (novel refinement): being **minimal** by generating as **few** explicit OIDs as possible, instead constructing virtual OIDs as query variable bindings (cf. Def. 6 – 8)
- Correctness of all transformations are proved (cf. Thm. 1 – 4)

Example KB/Queries Before Objectification

(Local constants prefixed by underscore; variables prefixed by question mark)

KB:

```
Document (
  Group (
    Forall ?Pers ?JobTitle ?Comp1 ?Comp2 (
      _transfer(?Pers ?Comp1 ?Comp2) :-
         And(_work(?Pers ?Comp1 ?JobTitle)
             _acquire(_buyer->?Comp2 _seller->?Comp1)))
    _e1#_transfer(_Tony _Rho4biz _Chi4corp _bonus->20000)
    _work(_Kate _Rho4biz "Director")
    _a1#_acquire(_buyer->_Chi4corp _seller->_Rho4biz)
  )
)
```

Queries:

```
   _work(?P ?C ?J)
?O#_work(?P ?C ?J)
   _transfer(?P ?C1 ?C2)
?O#_transfer(?P ?C1 ?C2)
```

Undifferentiated Static Objectification of KB/queries

Replace all oidless atoms with their existential forms

Objectified KB:

```
Document{
  Group (
    Forall ?Pers ?JobTitle ?Comp1 ?Comp2 (
       Exists ?1 (?1#_transfer(?Pers ?Comp1 ?Comp2)) :-
         And(Exists ?2 (?2#_work(?Pers ?Comp1 ?JobTitle))
             Exists ?3 (?3#_acquire(_buyer->?Comp2
                                    _seller->?Comp1)))
    )
    _e1#_transfer(_Tony _Rho4biz _Chi4corp _bonus->20000)
    Exists ?1 (?1#_work(_Kate _Rho4biz "Director"))
    _a1#_acquire(_buyer->_Chi4corp _seller->_Rho4biz)
  )
)
```

Objectified Queries:

```
Exists ?1 (?1#_work(?P ?C ?J))
?O#_work(?P ?C ?J)
Exists ?1 (?1#_transfer(?P ?C1 ?C2))
?O#_transfer(?P ?C1 ?C2)
```

Differentiated Static Objectification of KB/queries

- Replace each oidless atom `f(...)` with the following objectified form according to its occurrence
  - Ground fact:
    `_i#f(...)`, `_i` is a **new constant**, `i` $= 1, 2, ...$
  - Non-ground fact, rule conclusion atom, or query atom:
    `Exists ?j (?j#f(...))`, where `?j` is a **fresh variable** scoped universally by the enclosing rule
    (For queries, `?j` is a query variable encapsulated in an existential scope so that the bindings will not be returned)
  - Rule premise atom:
    `?j#f(...)`, where `?j` is a **fresh variable** scoped universally by the enclosing rule

Differentiated Static Objectification – Example

Objectified KB:

```
Document (
  Group (
    Forall ?Pers ?JobTitle ?Comp1 ?Comp2 ?2 ?3 (
      Exists ?1 (?1#_transfer(?Pers ?Comp1 ?Comp2)) :-
         And(?2#_work(?Pers ?Comp1 ?JobTitle)
             ?3#_acquire(_buyer->?Comp2 _seller->?Comp1))
    )
    _e1#_transfer(_Tony _Rho4biz _Chi4corp _bonus->20000)
    _1#_work(_Kate _Rho4biz "Director")
    _a1#_acquire(_buyer->_Chi4corp _seller->_Rho4biz)
  )
)
```

Objectified Queries:

```
Exists ?1 (?1#_work(?P ?C ?J))
?O#_work(?P ?C ?J)
Exists ?1 (?1#_transfer(?P ?C1 ?C2))
?O#_transfer(?P ?C1 ?C2)
```

Static/Dynamic Objectification of KB/queries

- Partition the set of KB predicates into two disjoint subsets: **non-relational** (at least one occurrence in a multi-tuple, oidful, or slotted atom, or in a subclass formula) and **relational** (no such occurrence) (cf. Def. 5)
- Statically generate OIDs only for the KB's oidless atoms with non-relational predicates (can be either undifferentiated or differentiated)
- Dynamically perform OID virtualization for query atoms using the KB's relational predicates with OID variables via equalities that unify an OID variable with an OID-constructor ('`_oidcons`') function application
- Make better use of the underlying Prolog engine in the PSOATransRun implementation for efficient inference on KB clauses with relational predicates

Dynamic Objectification of Atoms

- Leave all **KB atoms** with relational predicates unchanged
- For each **query atom** ω using a relational predicate `f` in KB ϕ, transformation $\text{obj}_d(\phi, \omega)$ includes these main cases:
  - If ω is a relationship, $\text{obj}_d(\phi, \omega) = \omega$
  - If ω has a **non-variable** (e.g., constant or expression) OID or a slot, $\text{obj}_d(\phi, \omega)$ is explicit falsity, here encoded as `Or()`
  - If ω has an OID **variable** and *m* tuples, being of the form `?O#f(`$[t_{1,1} \ \ldots \ t_{1,n_1}] \ \ldots \ [t_{m,1} \ \ldots \ t_{m,n_m}]$`)`, equivalent to a tupribution-like conjunction, copying `?O#f`, `And(?O#f(`$t_{1,1} \ \ldots \ t_{1,n_1}$`)` $\ldots$ `?O#f(`$t_{m,1} \ \ldots \ t_{m,n_m}$`)`, $\text{obj}_d(\phi, \omega)$ is a relational conjunction using equality

    `And(f(`$t_{1,1} \ \ldots \ t_{1,n_1}$`)` `?O = _oidcons(f` $t_{1,1} \ \ldots \ t_{1,n_1}$`)`
    $\ldots$
    `f(`$t_{m,1} \ \ldots \ t_{m,n_m}$`)` `?O = _oidcons(f` $t_{m,1} \ \ldots \ t_{m,n_m}$`))`
  - If ω is a membership of the form `?O#f`, $\text{obj}_d(\phi, \omega)$ is a disjunction shown in the following, where $n_1, \ldots, n_k$ are the *k* different arities of *f* in the KB:

$$\texttt{Or}(\text{obj}_d(\phi, \texttt{?O\#f(?X}_1 \ \ldots \ \texttt{?X}_{n_1}\texttt{)}) \ \ldots$$
$$\text{obj}_d(\phi, \texttt{?O\#f(?X}_1 \ \ldots \ \texttt{?X}_{n_k}\texttt{)}))$$

Static/Dynamic Objectification – Example

Use **differentiated** static objectification for atoms with non-relational predicates

Objectified KB:

```
Document (
  Group (
    Forall ?Pers ?JobTitle ?Comp1 ?Comp2 ?2 (
      Exists ?1 (?1#_transfer(?Pers ?Comp1 ?Comp2)) :-
         And(_work(?Pers ?Comp1 ?JobTitle)
             ?2#_acquire(_buyer->?Comp2 _seller->?Comp1))
    )
    _e1#_transfer(_Tony _Rho4biz _Chi4corp _bonus->20000)
    _work(_Kate _Rho4biz "Director")
    _a1#_acquire(_buyer->_Chi4corp _seller->_Rho4biz)
  )
)
```

Objectified Queries:

```
_work(?P ?C ?J)
And(_work(?P ?C ?J) ?O=_oidcons(_work ?P ?C ?J))
Exists ?1 (?1#_transfer(?P ?C1 ?C2))
?O#_transfer(?P ?C1 ?C2)
```

Outline

Unnesting

- Embedded psoa atoms
  - Widely used in object-centered languages such as RDF, N3, and Flora-2/F-logic as a shorthand notation
  - PSOA RuleML supports the use of embedded oidful atoms, e.g. `o1#c(p->f(o2#d))`
- Unnesting transformation
  - Decomposes nested atomic formulas into equivalent conjunctions

$$\text{Unnest}(\texttt{o1\#c(p->f(o2\#d))})$$
$$= \texttt{And(o2\#d o1\#c(p->f(o2)))}$$

  - Being maximal in that it can recursively extract oidful atoms – leaving behind their OIDs – not only from other atoms but also from expressions, which may themselves be embedded at any level

Unnesting – Formal Definition

$$\mathsf{Unnest}(\alpha) ::= \texttt{And}\,(\sigma_1\ \ldots\ \sigma_n)$$
$$\text{s.t. } \{\sigma_1, \ldots, \sigma_n\} = \cup_{t \in \mathsf{Parts}(\alpha)} \mathsf{Atoms}(t) \bigcup \{\mathsf{Trim}(\alpha)\}$$

$$\mathsf{Parts}(t) ::= \text{The set of top-level components of an atomic formula or a term } t$$

$$\mathsf{Atoms}(t) ::= \begin{cases} \emptyset & t \text{ is a simple term} \\ \cup_{s \in \mathsf{Parts}(t)} \mathsf{Atoms}(s) & t \text{ is oidless (expression)} \\ \cup_{s \in \mathsf{Parts}(t)} \mathsf{Atoms}(s) \bigcup \{\mathsf{Trim}(t)\} & t \text{ is oidful (atom)} \end{cases}$$

$$\mathsf{Trim}(t) ::= \text{Term/Formula obtained by replacing}$$
$$\text{every } s \in \mathsf{Parts}(t) \text{ in } t \text{ with } \mathsf{Retain}(s)$$

$$\mathsf{Retain}(t) ::= \begin{cases} t & t \text{ is a simple term} \\ \mathsf{Trim}(t) & t \text{ is oidless (expression)} \\ \mathsf{Retain}(\mathsf{Oid}(t)) & t \text{ is oidful (atom)} \end{cases}$$

Unnesting – Definition Explanation

- Unnest(α) is a conjunction of formulas σ_i without embedded atoms
- Each σ_i is a trimmed version of the top-level formula α or of some embedded psoa atom
- Atoms(t) contains each σ_i trimmed from an atom embedded in t or t itself. The set is constructed recursively.
- Trim(t) splits off all embedded atoms from t and leaves behind its 'ultimate' OID for each of them
- Retain(s) defines the left-behind term for each embedded term s

Unnesting – Example

Input α: `o1#c(p->f(o2#d))`

$$\{\sigma_1, \ldots, \sigma_n\}$$
$$= (\text{Atoms}(\texttt{o1}) \cup \text{Atoms}(\texttt{c}) \cup \text{Atoms}(\texttt{p}) \cup \text{Atoms}(\texttt{f(o2\#d)})) \bigcup \{\text{Trim}(\alpha)\}$$
$$= \text{Atoms}(\texttt{f(o2\#d)}) \bigcup \{\text{Trim}(\alpha)\}$$
$$= \text{Atoms}(\texttt{o2\#d}) \bigcup \{\text{Trim}(\alpha)\}$$
$$= \{\text{Trim}(\texttt{o2\#d}), \text{Trim}(\texttt{o1\#c(p->f(o2\#d))})\}$$

$\text{Trim}(\texttt{o2\#d}) = \text{Retain}(\texttt{o2})\#\text{Retain}(\texttt{d}) = \texttt{o2\#d}$

$\text{Trim}(\texttt{o1\#c(p->f(o2\#d))})$

$$= \text{Retain}(\texttt{o1})\#\text{Retain}(\texttt{c})(\text{Retain}(\texttt{p})\text{->}\text{Retain}(\texttt{f(o2\#d)}))$$
$$= \texttt{o1\#c(p->}\text{Retain}(\texttt{f})(\text{Retain}(\texttt{o2\#d}))\texttt{)}$$
$$= \texttt{o1\#c(p->f(}\text{Retain}(\text{Oid}(\texttt{o2\#d}))\texttt{))} = \texttt{o1\#c(p->f(o2))}$$

Output:

$\text{Unnest}(\texttt{o1\#c(p->f(o2\#d))}) = \texttt{And(o2\#d o1\#c(p->f(o2)))}$

Outline

Conclusions

- Refined PSOA semantics to allow a direct interpretation of oidless psoa terms, which includes the objectification restriction to establish the equivalence between an oidless atom and its existentially objectified form
- Formally defined systematics of objectification transformations and proved correctness
- Introduced a novel static/dynamic objectification approach for KB/queries, which is minimal in that it generates as few explicit OIDs as possible, instead constructing virtual OIDs as query variable binding
- Formalized unnesting transformation to decompose nested atomic formulas into equivalent conjunctions, which is maximal in that it can recursively extract oidful atoms not only from other atoms but also from expressions
- Objectification and unnesting have been implemented in PSOATransRun 1.1

Future work

- Explore further optimizations for objectification
- Complement the unnesting transformation with a flattening transformation for extracting ‘active’ expressions (i.e. built-ins and equality-defined functions) from both atoms and expressions